CRD

COMPUTATIONAL RESEARCH DIVISION

Application Oriented Performance Characterization and Benchmarking

Erich Strohmaier
Future Technology Group
EStrohmaier@lbl.gov
http://ftg.lbl.gov

SC2003

Co-sponsored by DOE/OSC and NSA

Office of Science
U.S. DEPARTMENT OF ENERGY

Motivation

- Application performance is what we care about most.
- Using real applications for performance work can be very tedious.
- Synthetic benchmarks are much easier.
- But we generally don't understand how the performance of synthetic benchmarks relates to applications!
- Is there a methodology to create synthetic benchmarks, which capture the main performance effects of real applications?

Approach

- Select the main performance aspects of our codes.
  - (or what we believe they are).
- Develop a quantitative characterization for these performance aspects.
  - Avoid using any specific hardware models for this characterization as far as possible!
- Develop a synthetic scalable performance probe implementing these characteristics.

Approach cont.

- Test the usefulness of the characterization with a set of codes.
- If we succeed the synthetic benchmark performance can be used as approximation for the application.
- Initial focus is the performance influence of global data-access.

Outline

- Application Performance Characterization (Apex)
- Memory access probe (Apex-MAP)
- Test Kernels
- Results and Correlations
- Extensions

Application Performance Characterization (Apex)

- We ignore computational aspects (for now).
- We view data access as composed of one or multiple data access streams.
- We characterize data access streams independent of each other.
- We try to use as few streams as possible (one).

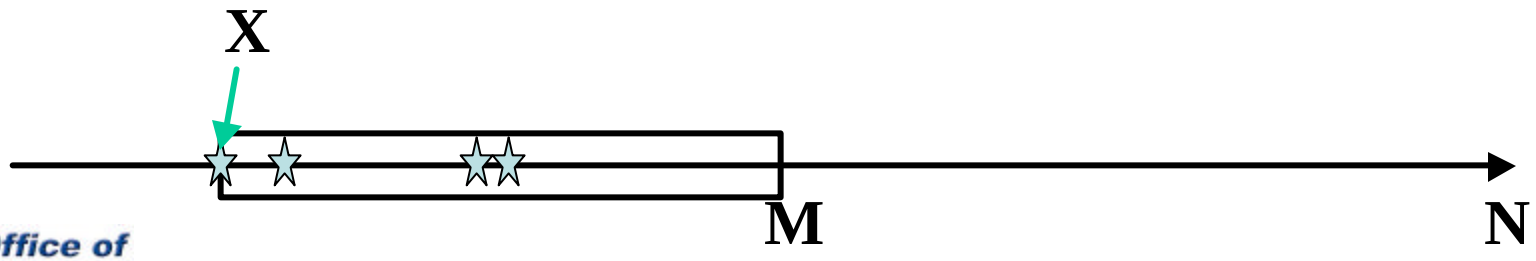
Performance Aspects of Data Access Streams

- **Regularity – 2 extremes:**
  - **Random walk in memory**
  - **Regular advance in memory**
- **Data set size (*M*)**
- **Length of contiguous data access (*L*)**
  - **Vector Length**
- **Temporal Locality (Re-use of data)**
  - **Characterized by the mean (*k*) of the number of accesses to the last location within the next *M* accesses.**
- **Stride (for regular access streams?)**

Re-Use Number

Define a "re-use" number:

- **Let *M* be the used memory in words.**
- **The code has a total of *N* data accesses.**
- **We look at all the accesses to a memory location *X* within the next *M* data access steps.**
- **We call the average *k* of all these access numbers the re-use number *k*.**



Synthetic Benchmark

- Now we design a synthetic benchmark which:
  - Generates a single data access stream.
  - Has our performance parameters as input.
  - Is scalable.
  - Is free of architecture specific performance artifacts.

Execution Model

- Use an array of size *M*.
- Access data in vectors of length *L*.
- Random:
  - Pick the start address of the vector randomly.
  - Use the properties of the random numbers to achieve a re-use number *k*.
- Regular:
  - Walk over consecutive (strided) vectors through memory.
  - Re-access each vector *k*-times.

Memory Access Probe Apex-MAP

For *random* access we choose:

- Use an 1024 address index for data access.
- Use the Power distribution for the non-uniform random address generator.
  - Self-similar and thus scale invariant.
  - Exponent ? in [0,1]
    - ?=1 : Uniform random access.
    - ?=0 : Access to a single vector only.

Apex-Map: Inner Loop - Random

```
for ( i = 0; i < N; i++) {
    initIndexArray(I);                          Initialize addresses
    CLOCK(time1);
    for (j = 0; j < I/4; j++) {                 Unrolled four times
        pos  = ind[j*4];
        …
        for (k = 0; k < L; k++) {               Vector access
            res0 += data[pos  + k];
            …
        }
    }
    CLOCK(time2);
}
```

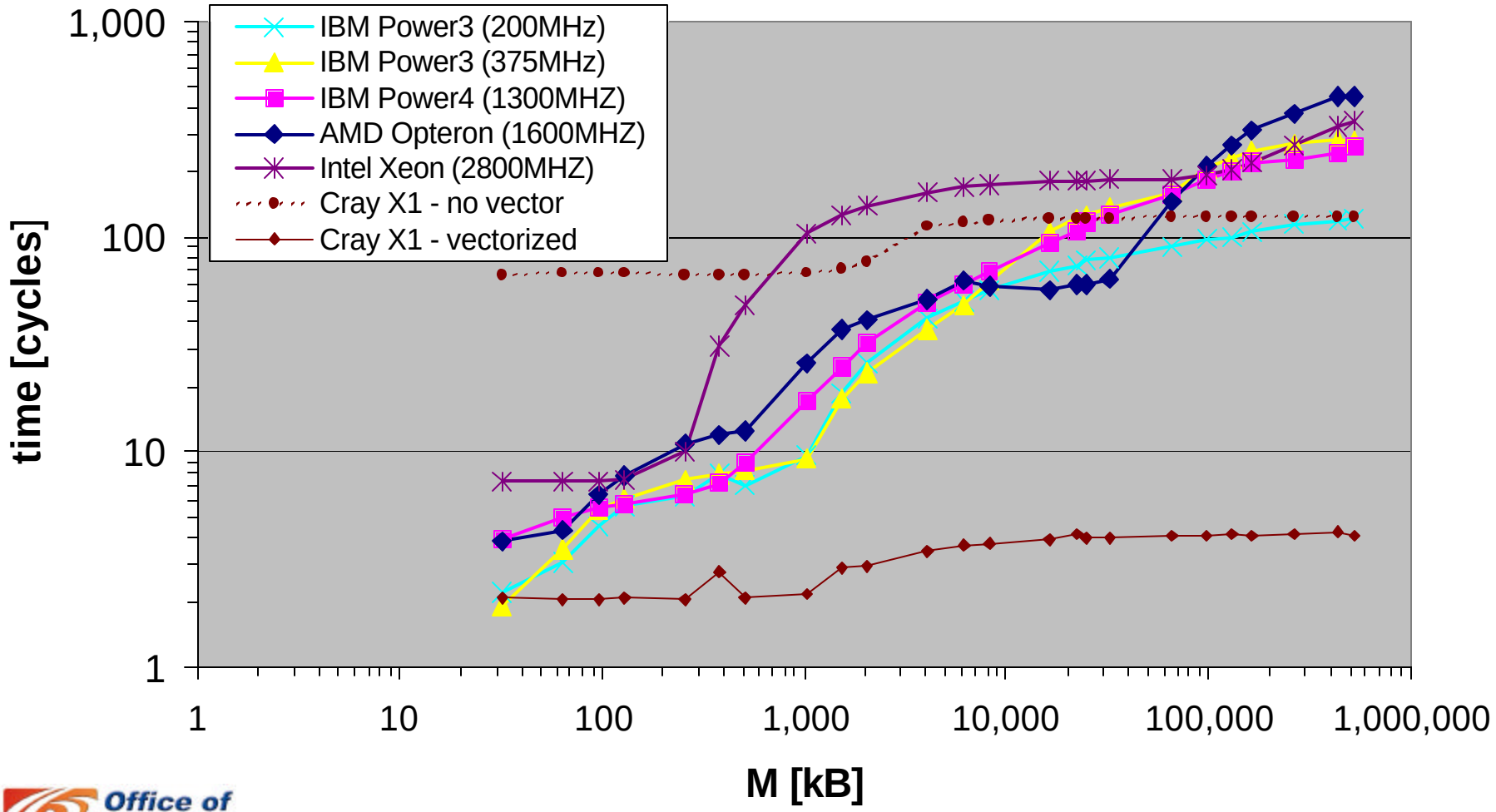
Office of Science
U.S. DEPARTMENT OF ENERGY

System Used

	Freq. [MHz]	L1 [kB]	L2 [MB]	Max outstand. load misses	Memory Bandwidth [GB/s]
Power3	200	64	4	4	3.2
Power3	375	64	8	4	4.0
Power4	1300	32	1.4 (L3 128)	8	10.6
Opteron	1600	64	1	32	5.3
Xeon	2800	8	0.5	4	3.2

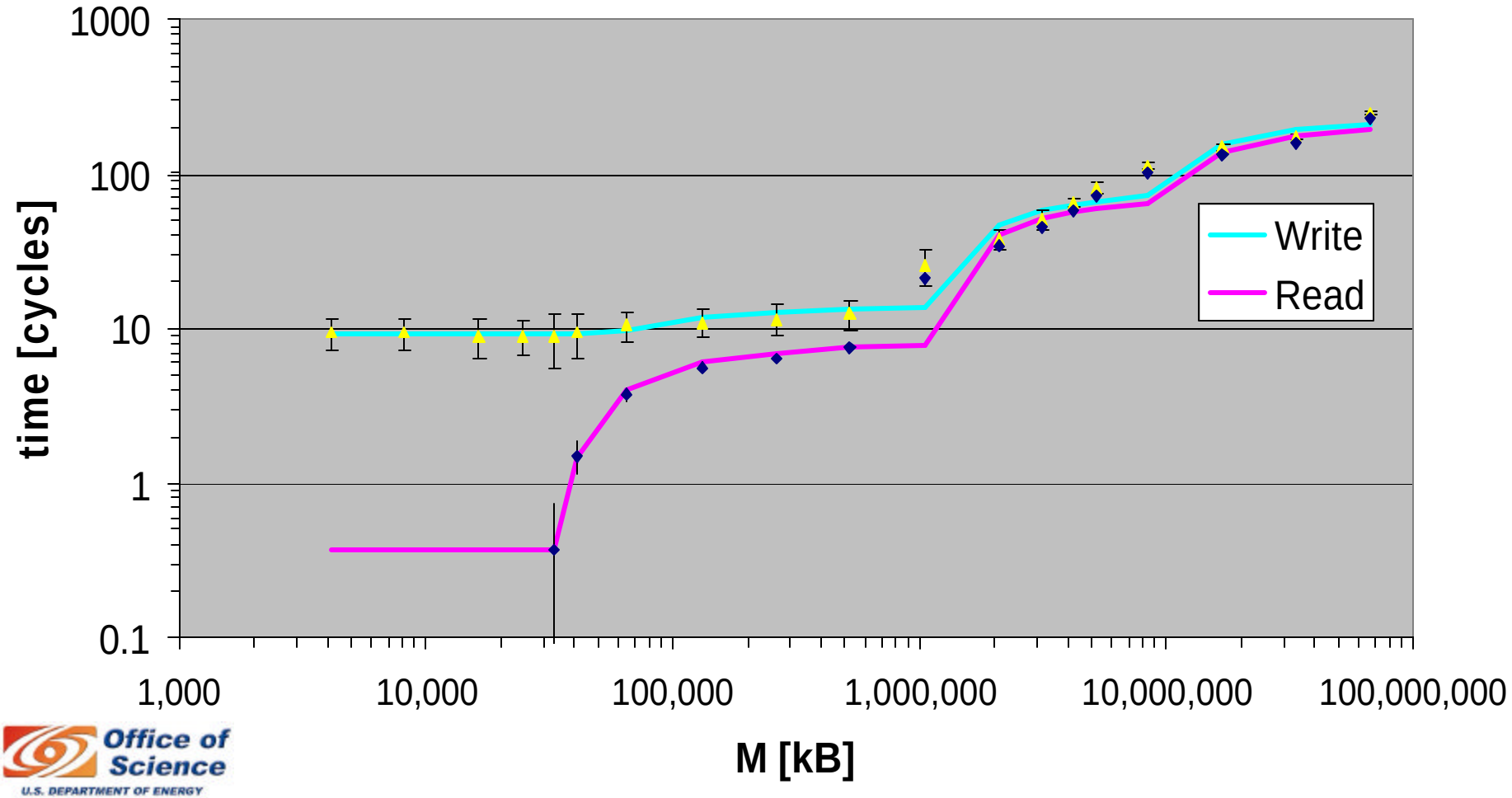
Memory Hierarchy Test

L=1; uniform random (a=1)

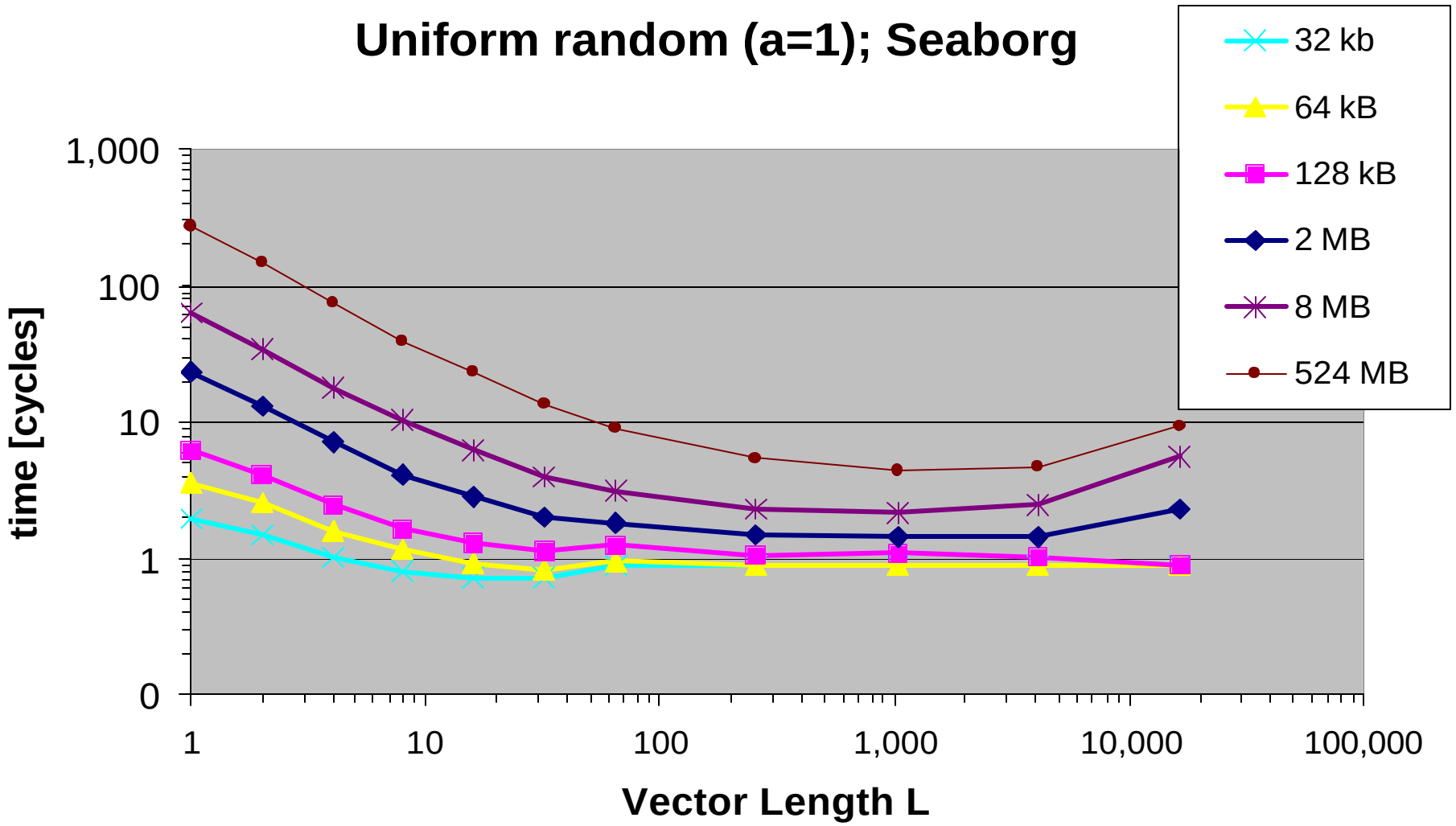


Memory Hierarchy Detail

L=1; uniform random (a=1); Seaborg



Vector Length



Validation - Ideal

- Take a set of test codes.
- Characterize them by selecting parameters.
- Time them.
- Run Apex-MAP with the same parameters.
- Check the correlation between the timings.

Validation - Real

- Use random access for all kernels to start with.
- Choose *M* to be the memory used by the kernel.
- Count the total number of memory accesses on a reference platform.
- Do a least-square fit of the times per access to determine a single set of *L* and *k* for all platforms.
  - 1 = L = 16k (powers of 2 or 4)
  - a ={0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1}

Test Kernels

- Radix (Integer Sort)
- N-Body (Interaction of N bodies in three dimensions)
- NAS CG (Conjugate Gradient, sparse linear systems)
- FFT (1-dimensional complex FFT)
- Matrix Transpose
- Matrix Matrix Multiplication
  - This is included as ‘worst’ (least fitting) test-case.

Statistical Details

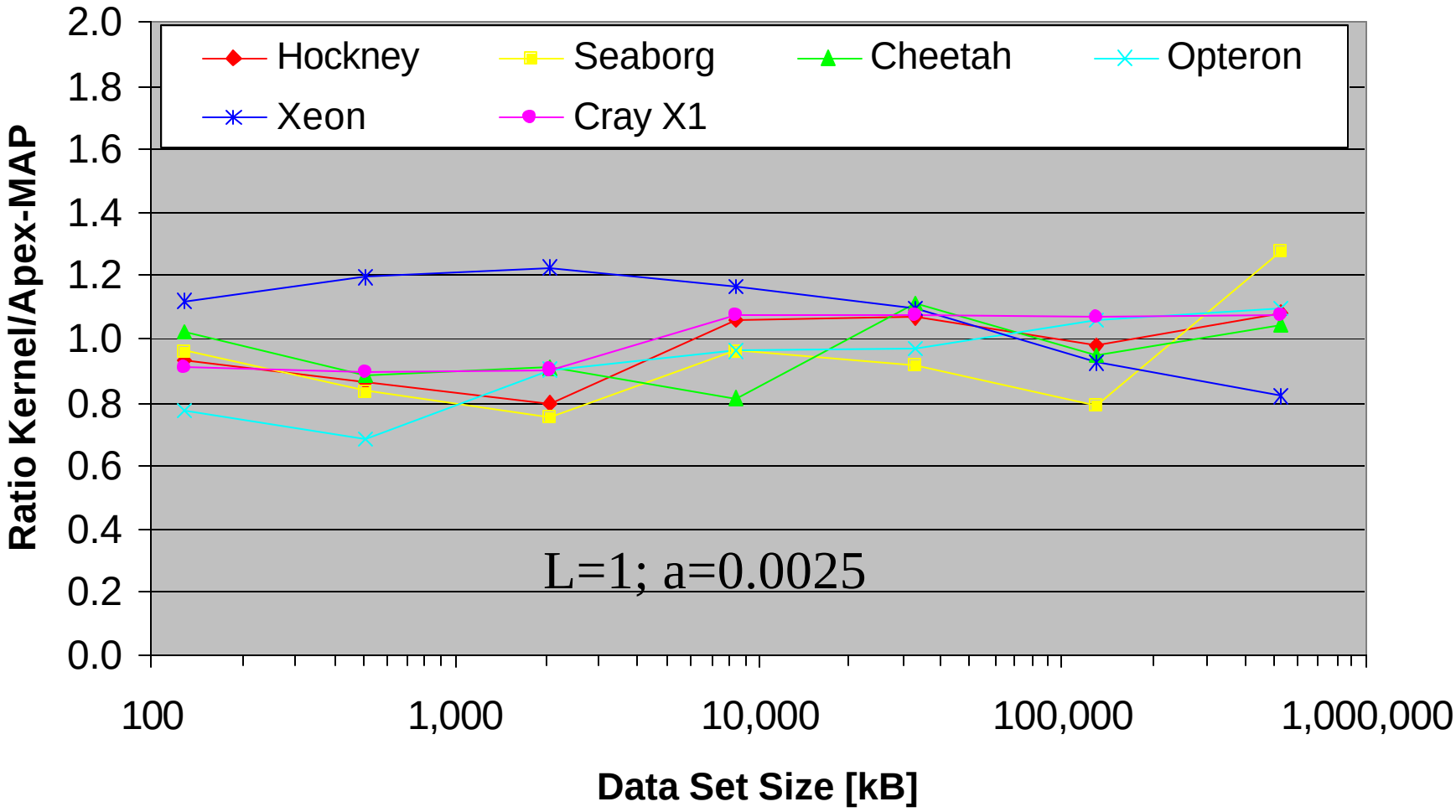
- **Linear Timing Model:**
  Kernel = Factor * Apex-MAP

- **No constant!**
- **Minimize the combined SSE (R^2).**
- **This does not always give a clear best choice.**

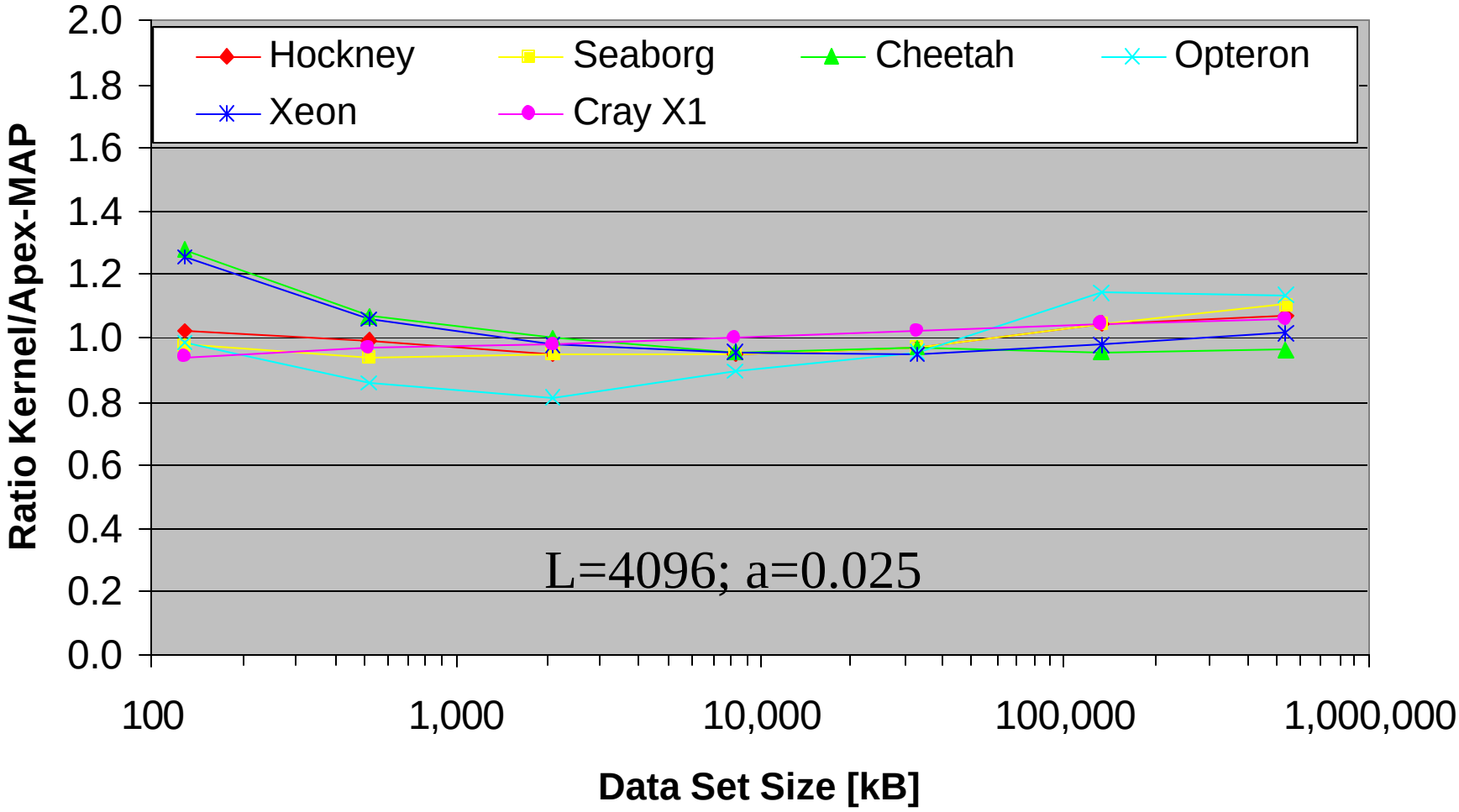
R^2 – Example
FFT on Seaborg

R-Square											
		a									
		0.001	0.003	0.005	0.010	0.025	0.050	0.100	0.250	0.500	1.000
	1	0.78	0.94	0.93	0.79	0.52	0.36	0.19	0.16	0.21	0.24
	2	0.62	0.88	0.96	0.85	0.61	0.38	0.27	0.25	0.26	0.28
	4	0.52	0.78	0.95	0.91	0.68	0.47	0.32	0.27	0.28	0.31
	8	0.37	0.64	0.89	0.98	0.81	0.59	0.40	0.33	0.33	0.36
	16	0.26	0.50	0.74	0.94	0.93	0.72	0.52	0.41	0.40	0.43
L	32	0.19	0.33	0.53	0.79	0.98	0.88	0.70	0.56	0.52	0.53
	64	0.10	0.22	0.36	0.60	0.90	0.97	0.90	0.76	0.70	0.69
	256	0.03	0.08	0.13	0.25	0.54	0.82	0.97	0.87	0.81	0.79
	1024	0.02	0.04	0.08	0.15	0.37	0.68	0.94	0.92	0.85	0.83
	4096	0.02	0.05	0.09	0.16	0.39	0.73	0.96	0.89	0.83	0.79
	16384	0.02	0.06	0.14	0.28	0.63	0.94	0.90	0.66	0.58	0.56

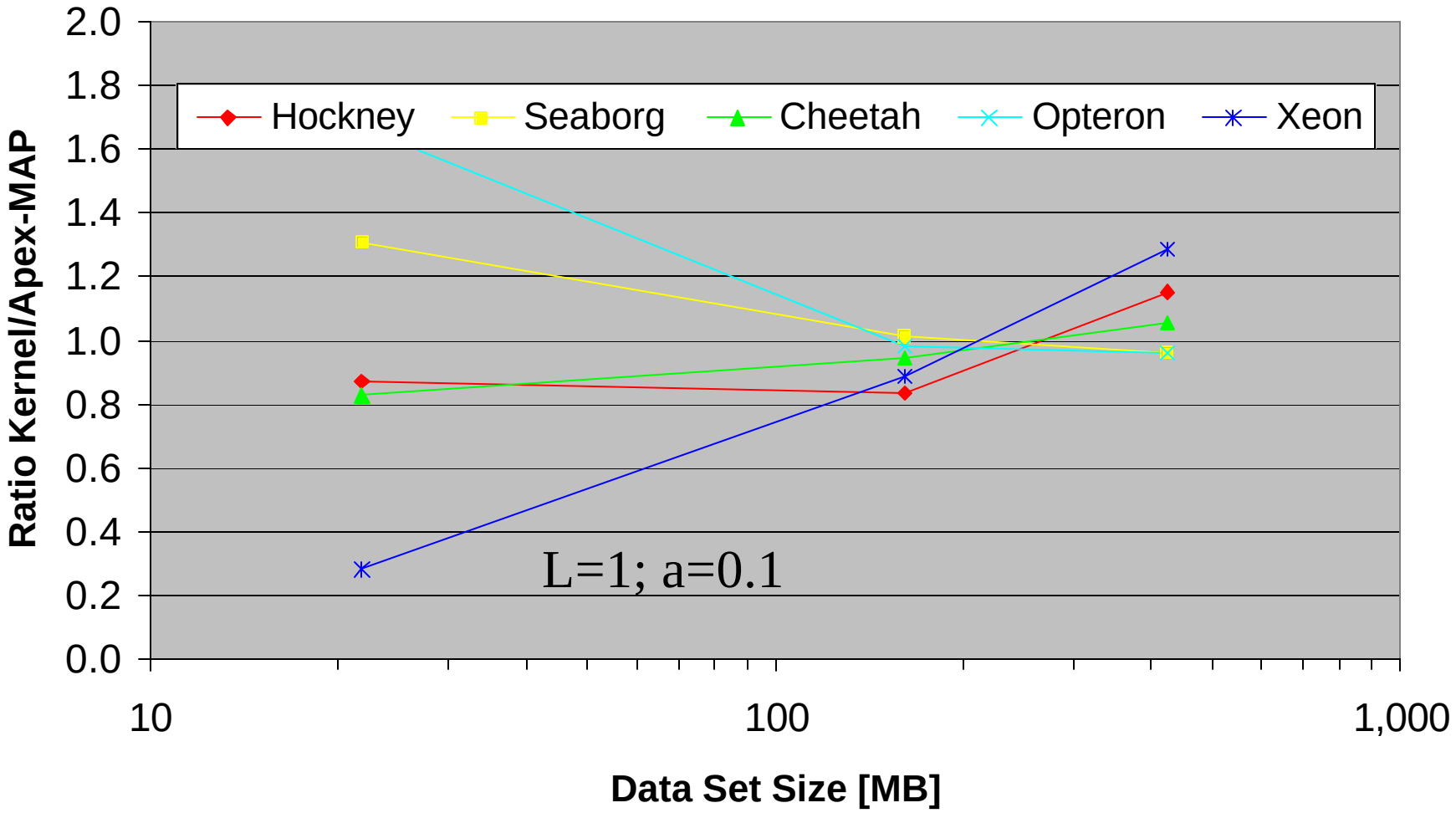
Radix


Hockney
Seaborg
Cheetah
Opteron
Xeon
Cray X1
Ratio Kernel/Apex-MAP
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
L=1; a=0.0025
100
1,000
10,000
100,000
1,000,000
Data Set Size [kB]

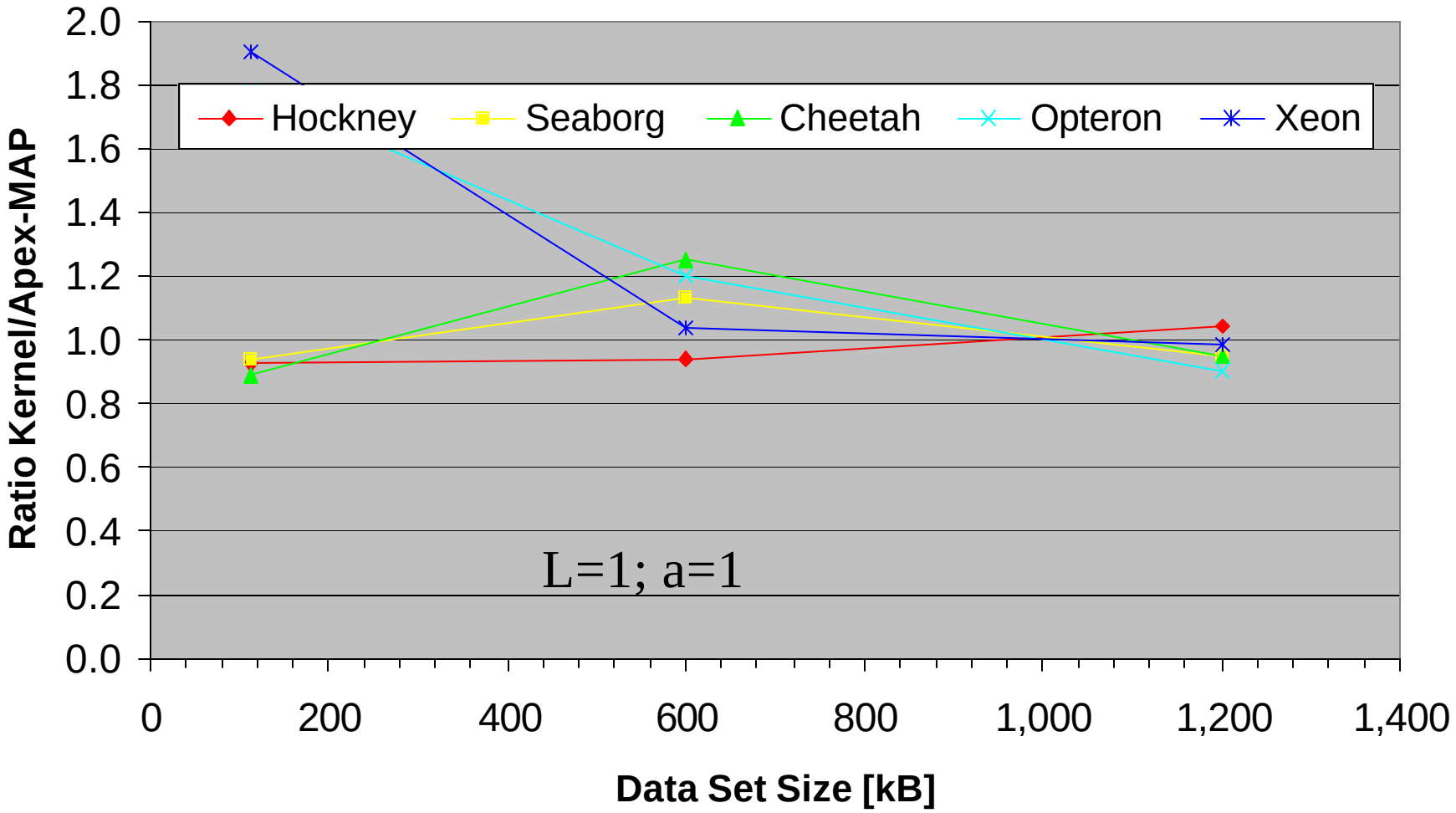
N-Body


Hockney
Seaborg
Cheetah
Opteron
Xeon
Cray X1
Ratio Kernel/Apex-MAP
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
L=4096; a=0.025
100
1,000
10,000
100,000
1,000,000
Data Set Size [kB]

NAS CG

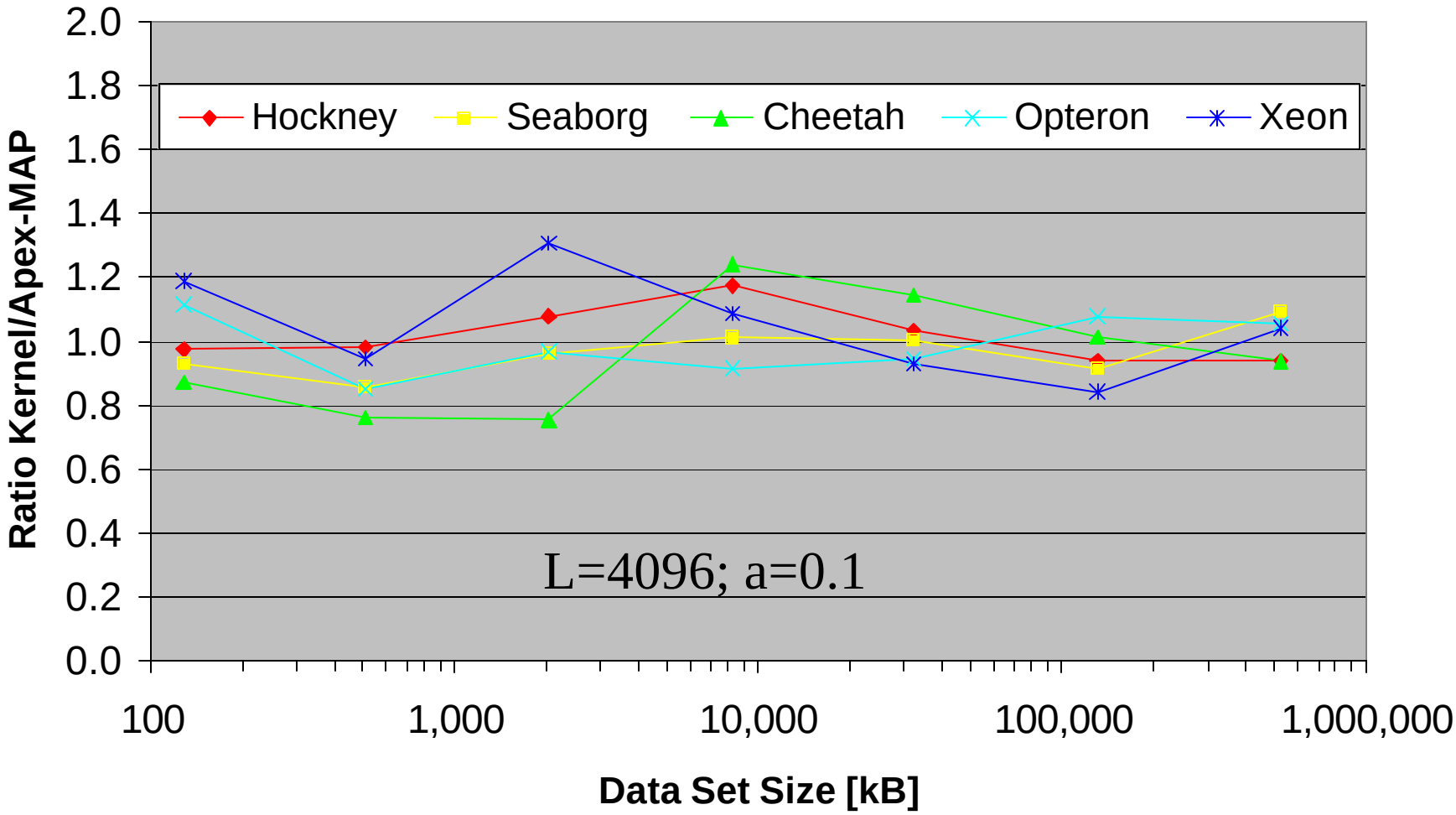

Hockney
Seaborg
Cheetah
Opteron
Xeon
Ratio Kernel/Apex-MAP
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
L=1; a=0.1
10
100
1,000
Data Set Size [MB]

CRD NAS CG – Indirect Vector only

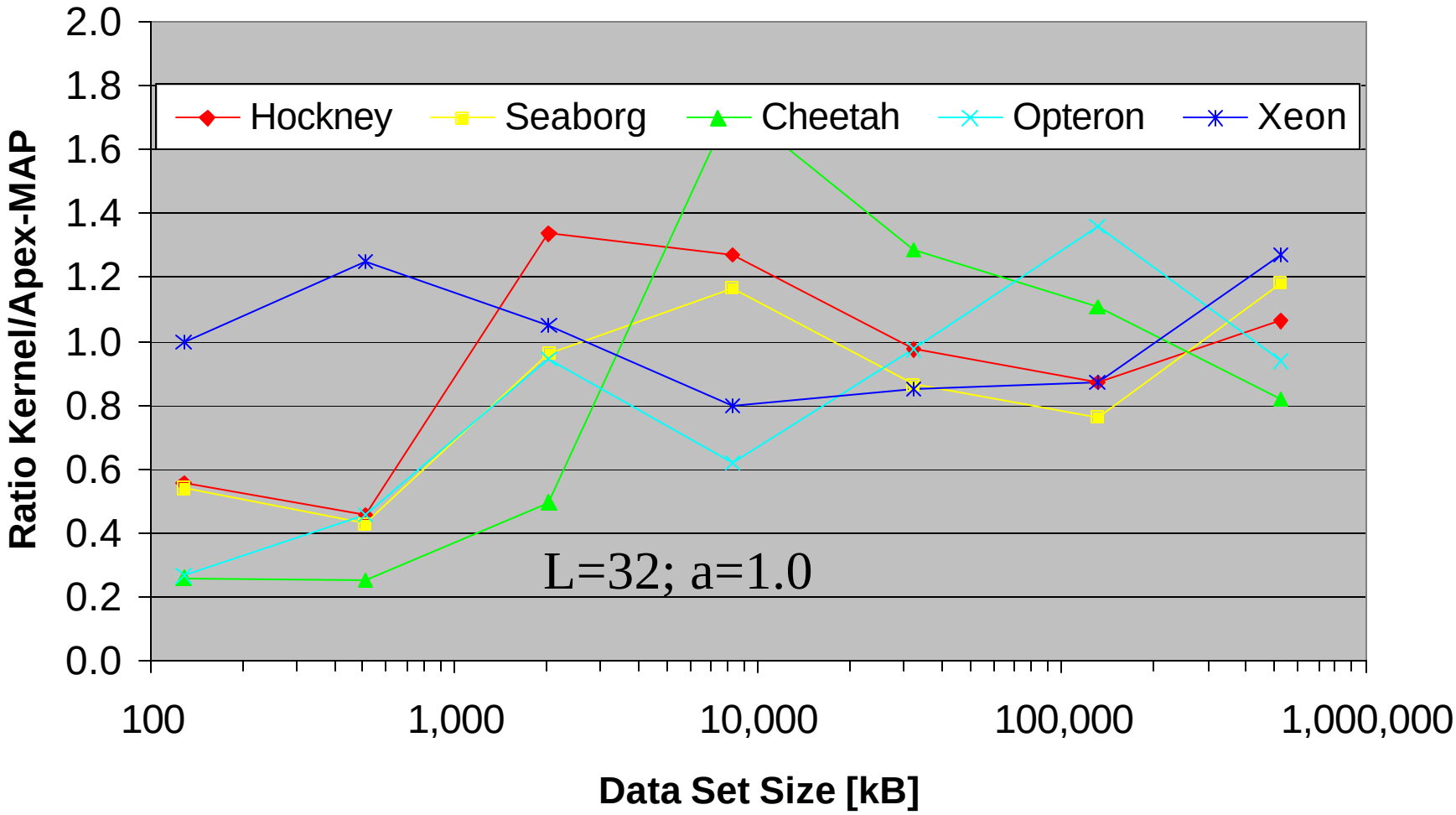


Office of Science
U.S. DEPARTMENT OF ENERGY

FFT

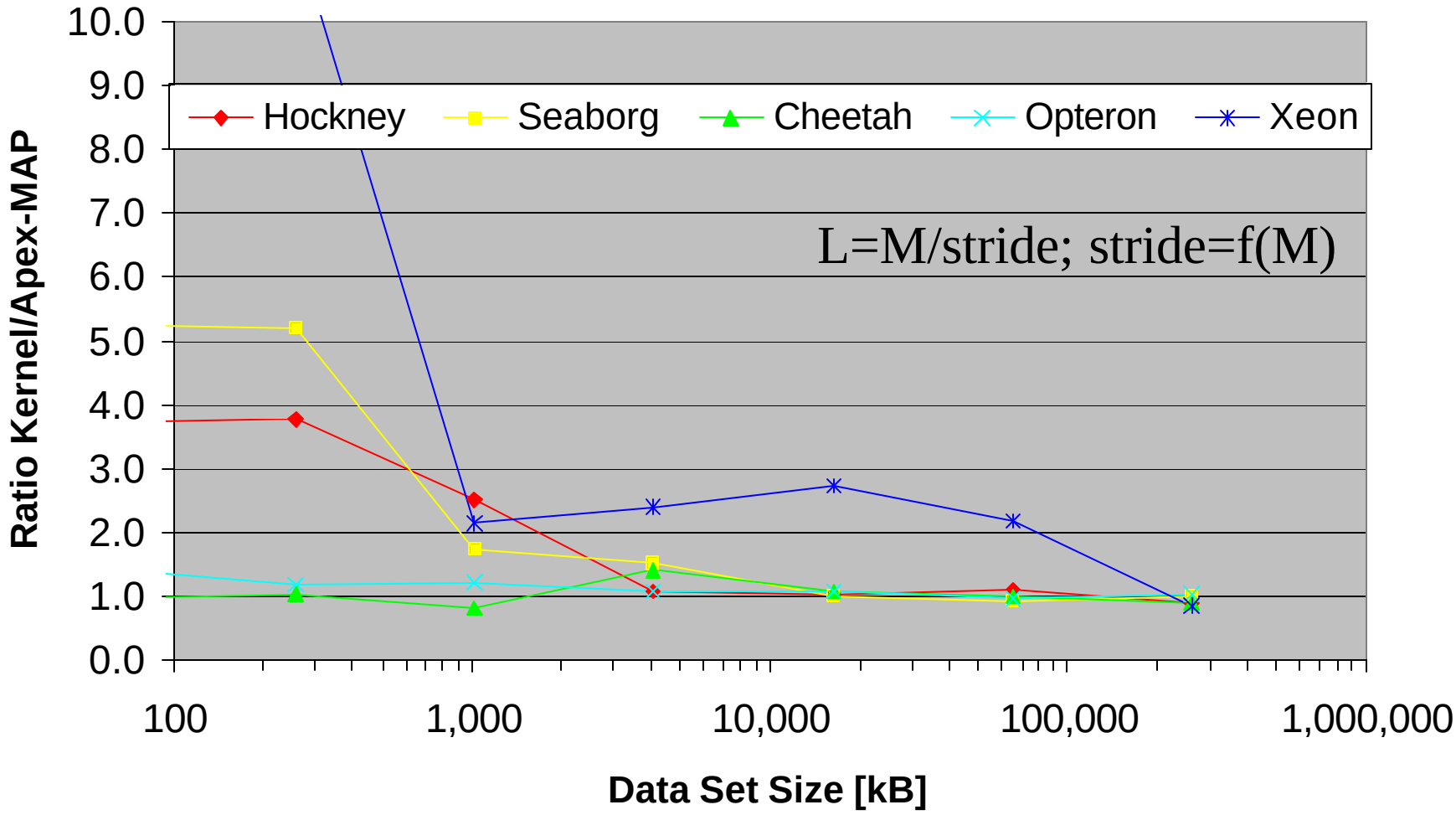

Ratio Kernel/Apex-MAP
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Hockney
Seaborg
Cheetah
Opteron
Xeon
L=4096; a=0.1
100
1,000
10,000
100,000
1,000,000
Data Set Size [kB]

Matrix Transpose



CRD

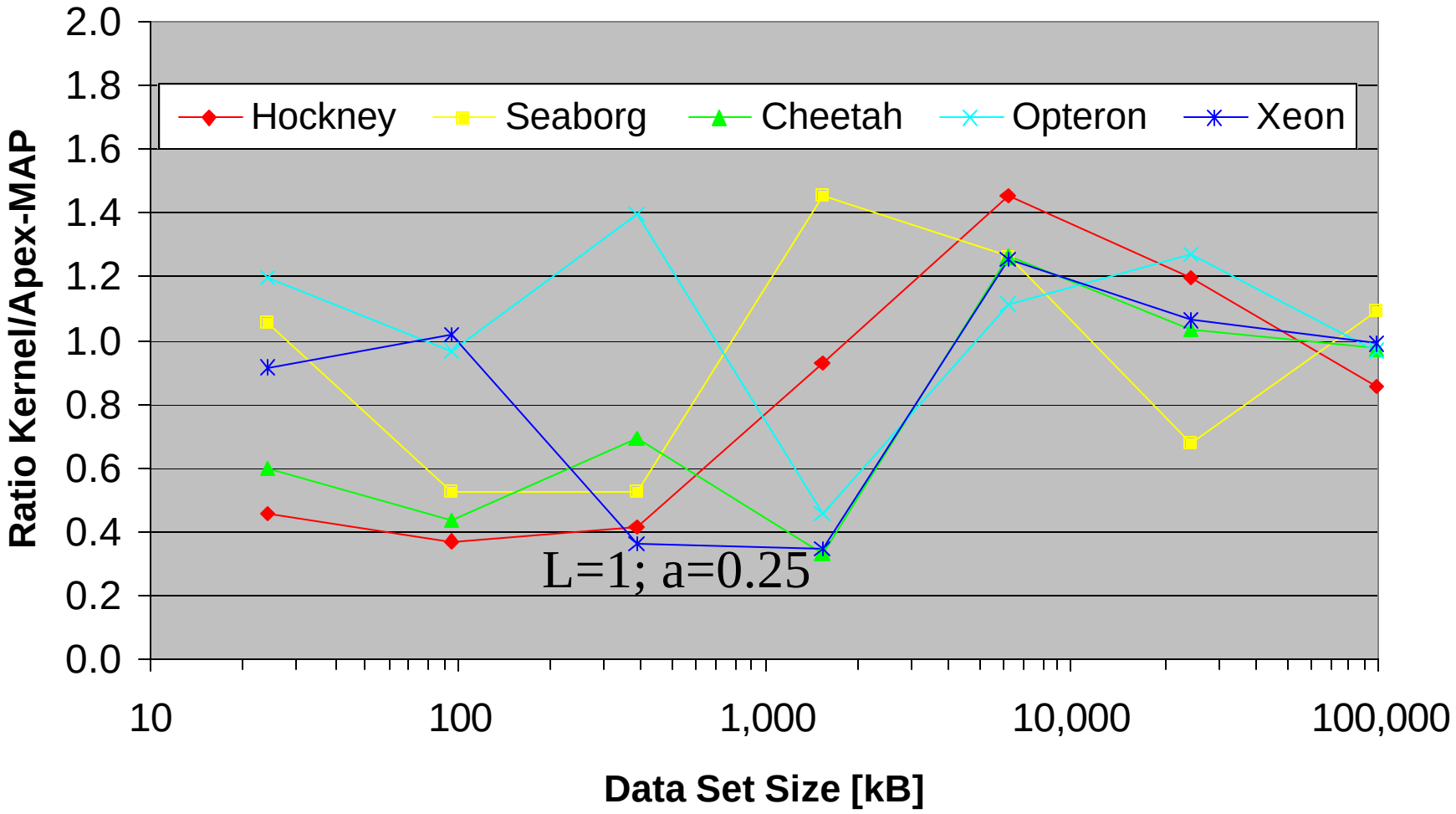
Matrix Transpose - regular



Office of Science
U.S. DEPARTMENT OF ENERGY

CRD

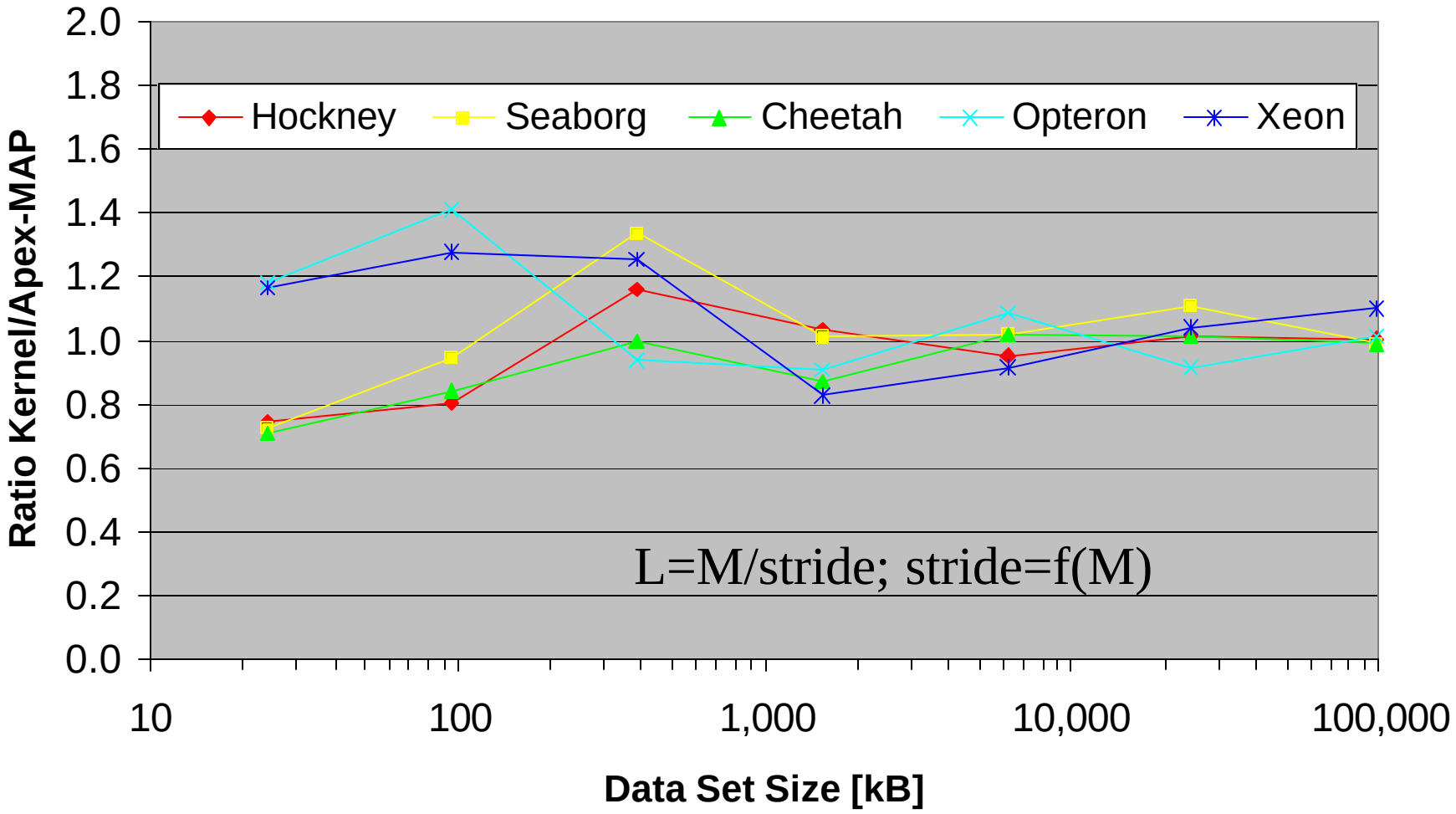
M-M Multiplication



Office of Science
U.S. DEPARTMENT OF ENERGY

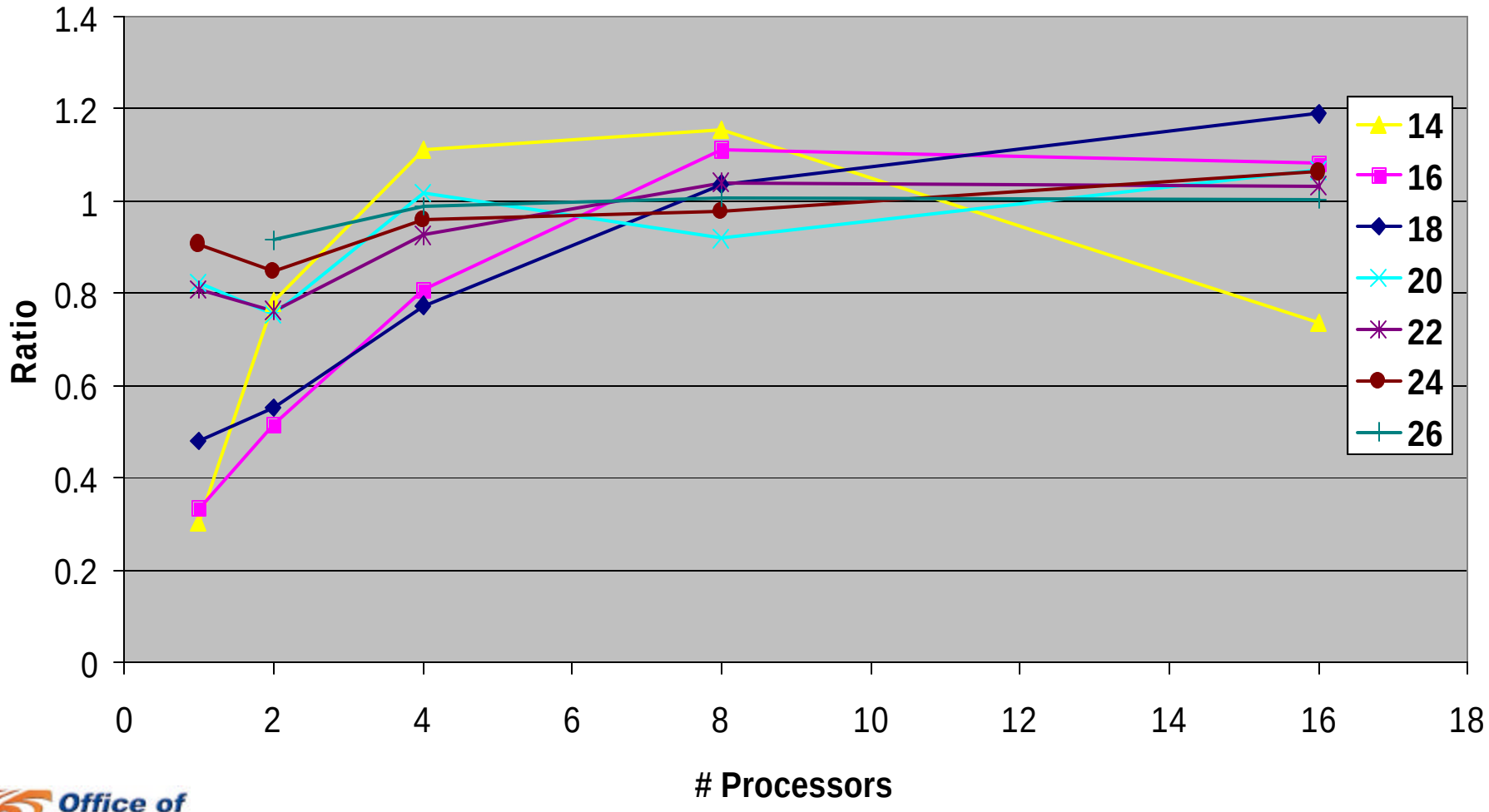
CRD

M-M Multiplication - regular



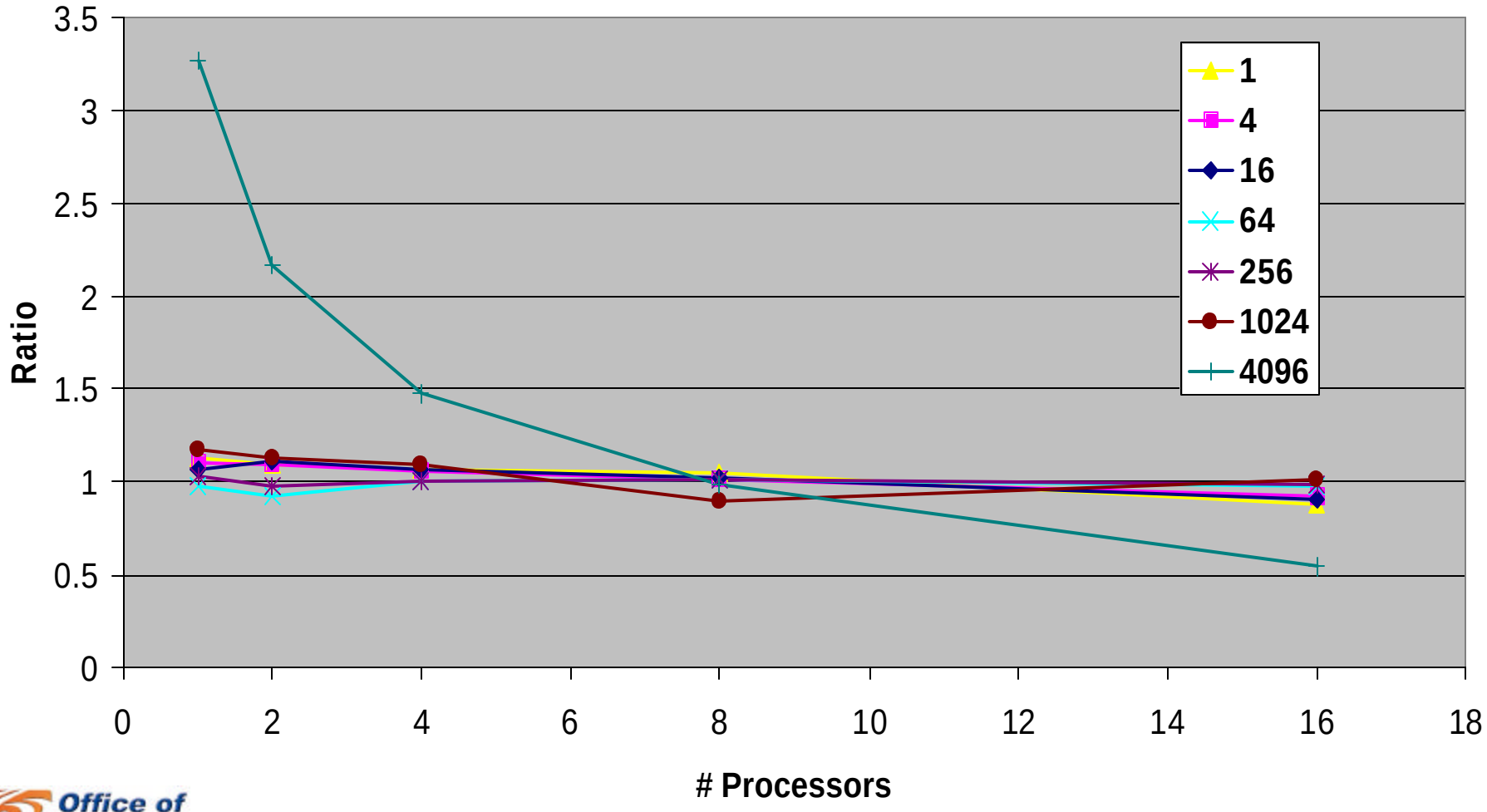
Office of Science
U.S. DEPARTMENT OF ENERGY

CRD Parallel Radix – Early Tests



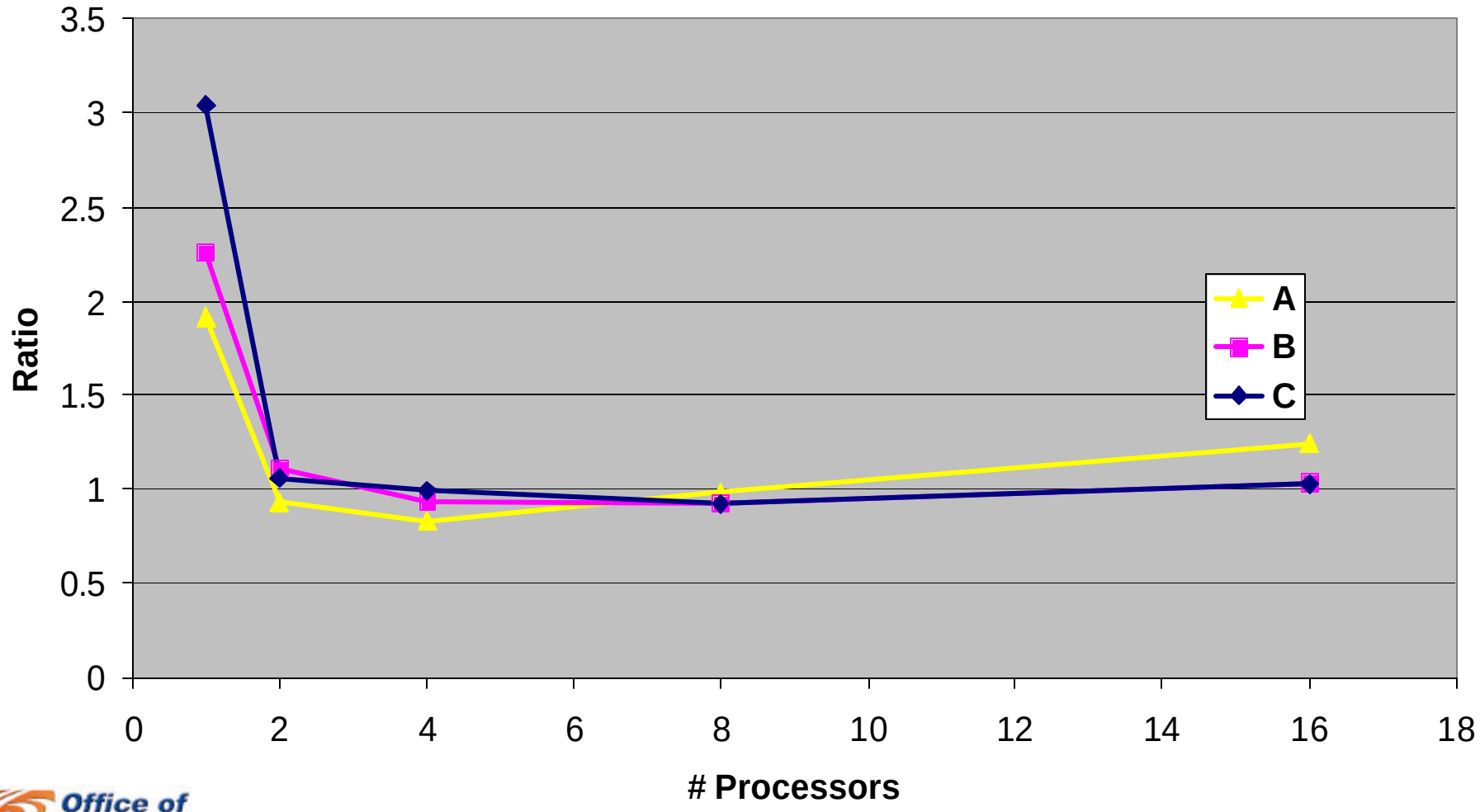
Office of Science
U.S. DEPARTMENT OF ENERGY

CRD Parallel N-Body – Early Tests



Office of Science
U.S. DEPARTMENT OF ENERGY

CRD Parallel CG – Early Tests



Office of Science
U.S. DEPARTMENT OF ENERGY

Experiences

- **A lot of the difficulties are in choosing the right details of the implementation.**
  - **Too many alternative implementations possible.**
  - **Which optimizations should one consider?**
- **The amount of measurements to do and to analyze becomes easily overwhelming.**
  - **Currently we take in the order of 3000 measurements per systems for the sequential study alone. (and we do it multiple times)**

Conclusions

- Approximation of the more irregular kernels by a single power function random access stream works fine.
  - In particular in the sequential case.
- The regular kernels are hard to approximate with random access streams and need a regular access stream.
- Backfitting *L* and *k* is not trivial.